Building Conversational Artifacts to Enable Digital Assistant for APIs and RPAs

Jayachandu Bandlamudi[1], Kushal Mukherjee[1], Prerna Agarwal[1], Ritwik Chaudhuri[1], Rakesh Pimplikar[1], Sampath Dechu[1], Alex Straley[2], Anbumunee Ponniah[2], Renuka Sindhgatta[1]

[1]IBM Research
[2]IBM Software

Abstract

In the realm of business automation, digital assistants/chatbots are emerging as the primary method for making automation software accessible to users in various business sectors. Access to automation primarily occurs through APIs and RPAs. To effectively convert APIs and RPAs into chatbots on a larger scale, it is crucial to establish an automated process for generating data and training models that can recognize user intentions, identify questions for conversational slot filling, and provide recommendations for subsequent actions. In this paper, we present a technique for enhancing and generating natural language conversational artifacts from API specifications using large language models (LLMs). The goal is to utilize LLMs in the "build" phase to assist humans in creating skills for digital assistants. As a result, the system doesn't need to rely on LLMs during conversations with business users, leading to efficient deployment. Experimental results highlight the effectiveness of our proposed approach. Our system is deployed in the IBM Watson Orchestrate product for general availability.

Introduction and Background

AI-driven automation is leading to the digital transformation of businesses in the fourth industrial revolution era (Laurent, Chollet, and Herzberg 2015). Automation systems with interactive AI capabilities help users to automate repetitive tasks and improve the performance of their business (Devarajan 2018). Application Programming Interface (API) services and to a greater extent RPAs (Robotic Process Automation) are the mainly adopted automation technologies for automation (Syed et al. 2020).

However, despite their omnipresence, there is an increasing demand to bring APIs and RPAs into conversational automation systems. Task-oriented conversational automation, such as chatbots and virtual assistants, represents the next frontier in user engagement and customer support expanding from basic Q&A type tasks to include actions (Yaghoub-Zadeh-Fard, Benatallah, and Zamanirad 2020). Integrating APIs into these systems opens up a world of possibilities, as it allows them to access real-time data, perform complex tasks, and connect with external services on demand. Given that many typical business users do not have the technical know-how in the area of software development and automation, Task-oriented conversational systems alleviate the main difficulty in the adoption of automation technologies.

One way to enhance the accessibility of automation technologies like RPA, especially unattended bots, is by encapsulating their API services within chatbots (Vaziri et al. 2017). Employing natural language for interactive automation can offer a more intuitive experience for business users, as conversational systems strive to communicate in a human-like manner. Traditionally, creating chatbots involves multiple steps including (Bocklisch et al. 2017; Sabharwal et al. 2020) i) training an intent classifier, which necessitates expertise in machine learning, natural language processing, and domain-specific data from experts, ii) Identification of the inputs and their data types (often called slots) required for invoking the API and, iii) processing of the result of the API and providing next action recommendations.

LLMs with vast parameter sizes, ranging from millions to billions, have demonstrated promising outcomes in various natural language tasks, including intent recognition and language generation (Zhao et al. 2023). These models are trained on trillions of tokens extracted from open-domain text sources. They have been used to enable a conversation for APIs (Patil et al. 2023) in a few cases. However, when it comes to enterprise automation, specialized domain knowledge is often required, which is not inherently present in the data used for training these LLMs (Yang, Uy, and Huang 2020). Consequently, adjustments or additional contextual information are necessary to make these models relevant to specific domains or prompts. Unfortunately, many businesses, especially small and medium enterprises, face challenges in fine-tuning such models due to limited resources or expertise, making the process prohibitively expensive. Further, LLMs suffer from the problem of uncertain outputs and hallucinations with limited control over what is generated (Bang et al. 2023). In addition to that, there is the issue of the high cost of deployment that requires a lot of infrastructure (including GPUs) to scale.

To alleviate the above issues, we have created a human-guided process of authoring and wrapping API services as Digital Assistant/chatbot skills. This process is called the *build* step. The objective is to use LLMs in the *build* phase to assist the human to author the skills for the digital assistant.

This is typically a one-time step for each automation. Subsequently, the system does not have to leverage LLMs while conversing with business users. Human-guided authoring of skills is a preferred approach until LLMs reach that level of maturity. It also provides a level of trust and indemnity to the client.

In this paper, we describe a method that leverages LLMs to generate conversational artifacts that can be used to train intent classifiers with reduced reliance on AI expertise, generate questions for conversational input slot filling, and derive the next actions for skills.

Our approach involves language generation using the meta-data from the OpenAPI [1] specifications of the APIs[2]. Harnessing the generative capabilities of LLMs offline during *build* step helps eliminate the run-time expenses associated with high-end hardware. Through our experiments, we demonstrate that we can achieve improved performance by using LLMs at *build* time for intent classification, question generation for slot-filling, and next-best agent prediction, while simultaneously lowering the barriers for non-AI experts by involving human subject matter experts. Further, our approach of using general-purpose instruct-tuned models ensures that the same model can work for multiple tasks and may be deployed efficiently.

The following are the contributions of the paper:

- Enabling a conversational interface for APIs/RPAs by the generation of natural language conversational artifacts in an agent-based framework that significantly reduces *build* effort.
- Enabling human-guided building of conversational interface for APIs/RPAs for improving trust and protecting against indemnity.
- Use of LLMs to generate data for intent classification, question generation for slot filling, and next action recommendations for invoking APIs/RPAs in conversations

System and Architecture

We aim to address the challenges mentioned above by enhancing accessibility for users to run automation (APIs/RPAs) and intelligent sequencing of automation. To achieve this goal, we employ a conversational interface that is accessible to automation users on proprietary platforms. These interfaces enable users to employ natural language to execute automation for addressing business-related tasks. In this paper, we introduce a comprehensive framework that allows most current APIs and RPAs to be seamlessly accessed and utilized through conversational interactions.

To wrap a conversation interface around an API, we make use of a framework for constructing digital assistants, as outlined in (Rizk et al. 2020). The authors introduced a framework designed to connect RPAs and corporate chatbots, employing a multi-agent orchestrator that facilitates interactive automation via natural language and AI planning. Next, we introduce the fundamental principles utilized within this framework as described in (Rizk et al. 2020).

[1]https://spec.openapis.org/oas/latest.html

[2]APIs refer to the OpenAPI specifications from here on

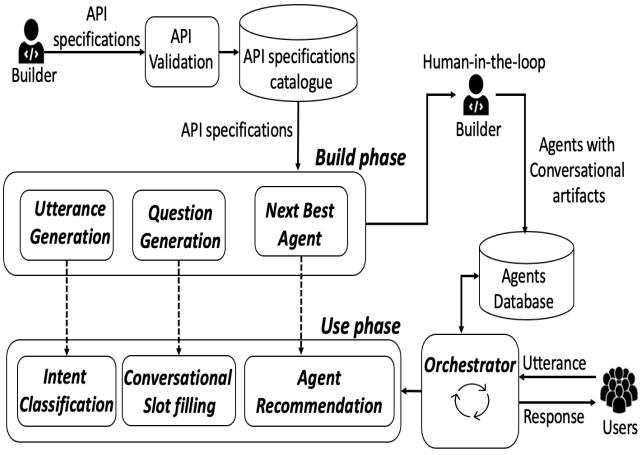

Figure 1: System Architecture

- ***Skills:*** are the atomic functions, there exist five types of skills in the framework: i) *understand* user's intention, ii) *request* information from the user to perform the action (conversational slot filling) iii) *act* on the request, and iv) *respond* to the user and v) provide recommendations for *NBA* (next best agent). In detail, the *understand* skill refers to the natural language understanding of a user's utterances. The *request* skill asks natural language questions to the user for gathering the required inputs. The *act* skill, in our context, is invoking the API/RPA with the gathered inputs. The *respond* skill is for providing a human-consumable response based on the output of the *act* skill. Finally, the *NBA* skill provides recommendations for the next best agents that the user may execute.
- ***Agents:*** are a composition of *understand-request-act-respond-NBA* skills that transform the set of skills into a conversational entity. Agents receive a user utterance and the current context of the execution or state. They act on the utterance based on the context and respond with an updated context and confidence in their response, indicating their comprehension of the input and response.
- ***Orchestrator:*** handles interaction between agents, that include *human* agents. The main function of an orchestrator is to i) select the correct agent from a predefined set of available agents to achieve a task, ii) manage the context, and iii) manage the flow of the conversation between the agents. The orchestrator receives a user utterance and forwards it to all agents available along with appropriate context from the user. Once it receives the agents' preview response, it chooses the most appropriate agent (or sequence of agents) according to user utterance. After selecting the agent, it connects with the agent to complete the execution and manage the context.

Thus, the process of wrapping an API operation in a conversational interface involves the authoring of the corresponding agent specification using API metadata. The agent specification requires a set of natural language conversational artifacts such as: (i) The *understand* skill requires the generation of multiple utterances based on the description of

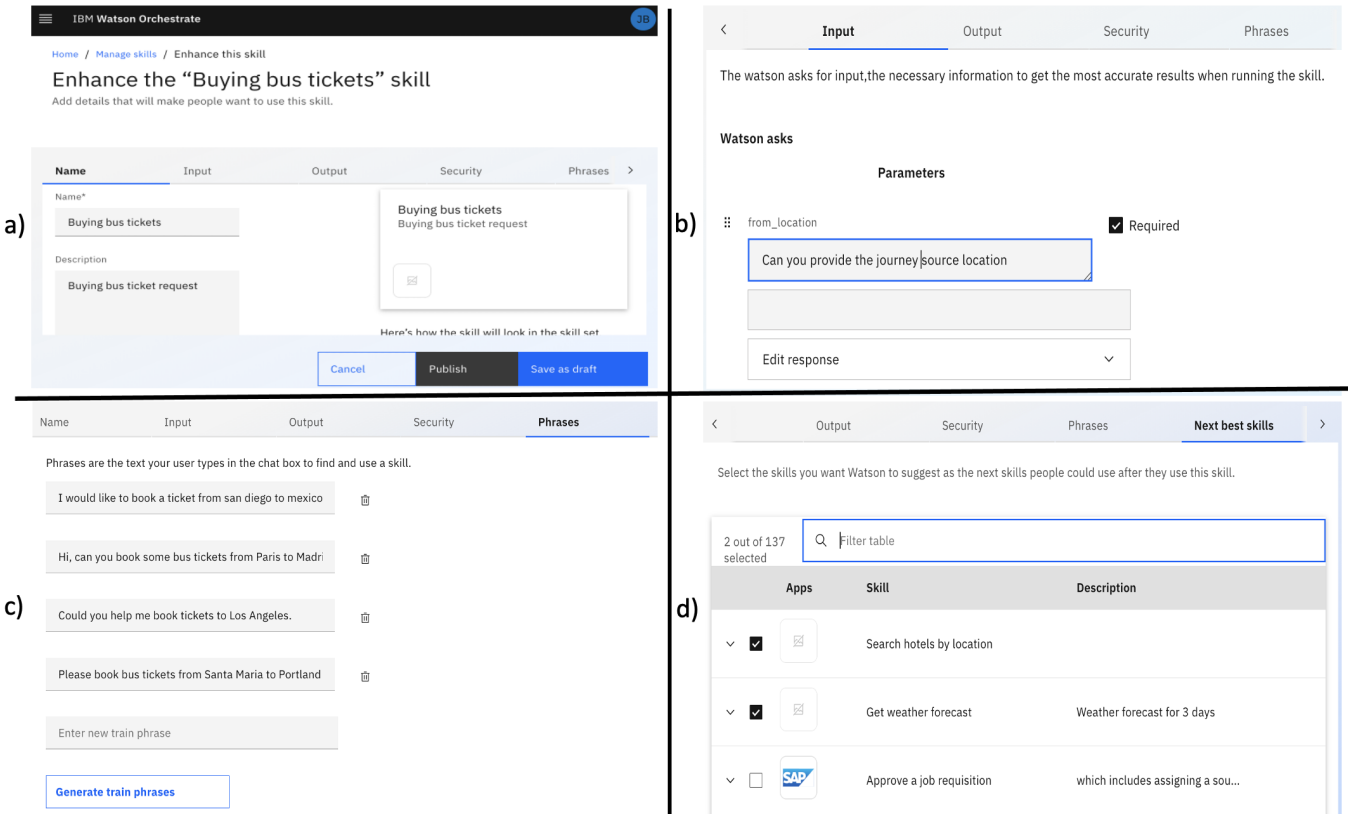

Figure 2: System Builder Interface. (a) The process of adding a new skill from an API specification (b) LLM generated questions are displayed for each attribute of the API call that the builder reviews/updates (c) Generated utterances that would be used to train the intent classifier (d) Next actions are displayed to be reviewed/updated by the builder

the API operations. These utterances are then used to train the intent classifier, (ii) The *request* skill requires the generation of contextual questions to request user information to execute the task. These questions have to be generated based on the API metadata including description and input-output schema, (iii) The *NBA* skill to obtain possible next agents from the available set of agents and also perform semantic mapping of inputs and outputs of agents to sequence them.

Our system architecture, described in Figure 1, shows two personas: one is the "builder" and the other is the "user". The "builder" persona selects what APIs or RPAs have to be exposed via the conversational interface, i.e., the creation of the agent. Whereas the "user" persona uses the APIs or RPAs via the conversational interface. When a "builder" adds a new API it is first validated and captured in a catalog and then the *build* phase is triggered. The three components of *build* phase are used to generate the required conversational artifacts of an agent specification. Once an agent is generated, the "builder" can review and update (if required) before publishing the agent. Figure 2 shows how a builder would choose a service and review/update the conversational artifacts

These agents with conversational artifacts are then stored in the agent database. Figure 3 shows a sample agent specification with the conversational artifacts marked in red color.

Once the agents are published, they are available to the "users" of the system and invoked through a user utterance. The orchestrator (Rizk et al. 2020) receives an utterance from the user and invokes the *understand* skill of the agents. The *understand* skill classifies the utterance to one of the agents. In cases where the classifier confidence is low or there are multiple agents with similar confidences, an appropriate fallback or disambiguation action is taken (Rizk et al. 2020). Once the correct agent is selected, the orchestrator executes the sequence of skills *request-act-respond-NBA*. Figure 4 shows the "users" conversational experience. If the orchestrator selects an incorrect agent, then user can cancel the execution of agent. During *use* time, there is no need for expensive inference on LLMs as all the conversational artifacts have been pre-generated and reviewed in the *build* phase.

Utterance Generation

The Orchestrator leverages the *understand* skill of agents to identify the correct agent according to the provided user utterance. The *understand* skill consists of an intent classifier trained in the *build phase* of the corresponding agent as shown in Figure 3. During the *build* phase we obtain the agent description from the corresponding API operation and it is to be noted that only one description is provided for

```
_id: buying_bus_ticket_agent_1.0
actuator:
    pipeline:
    -   args:                                              Request Skill
            authentication: None
            operation: conversational_slot_filling
            payload: '{$._event_data.submitted_form}'
        user_slots: [{"attribute":"to_location",
                      "question":["what is your destination",
                                  "to which city are you travelling"]},...]
        skill: orchestrator.skills.conversational_slots
    -   args:                                              Act Skill
            authentication: basic
            credentials:
                password: '{$USER_PASSWORD}'
                username: '{$USER_NAME}'
            endpoint: '{$SERVICE_URL}/api/v2/bus/buy_tickets/'
            operation: post
            payload: '{$._event_data}'
        skill: orchestrator.skills.execute
    -   args:                                              Respond Skill
            endpoint: '{$SERVICE_CALLBACK_URL}'
            operation: post
            payload:
                booking_status: '{$._event_data.booking_status}'
        next_best_agent: [ hotel_reservation_agent_1.0 ,   NBA Skill
                           check_the_local_events_1.0 ]
        skill: orchestrator.skills.execute
evaluator:
    pipeline:
    -   args:                                              Understand Skill
            client_deployment_config: _import
            workspace_id: cd75826f-f319-48bb-b0e1-b5fbdf137b7f
        training_utterances: ['I want to buy bus tickets',
                              'Book a bus ticket to LA',...]
        assign_to: wa_skill
        skill: orchestrator.intent_classifier
```

Figure 3: Example agent with skills and the generated conversational artifacts marked in red

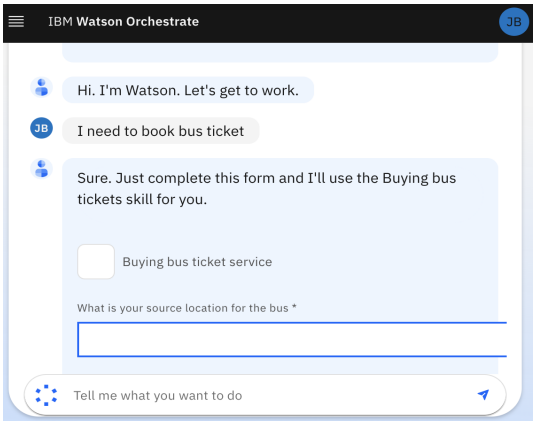

Figure 4: System User Persona Interface

each agent. Since a single description as the training sample is not sufficient to train the intent classifier, the training data is augmented via utterance generation.

As shown in Figure 5 utterance generation involves several steps. At first, we obtain the agent description from the API specification. Using the agent description and an LLM prompt as input, we generate more natural language utterances for the description. Additionally, outliers are detected and removed. It is followed by the diversity selection to ensure a diverse set of generated utterances. The filtered set of utterances is used for training the intent classifier. The details for each step are provided below,

Step-1 (LLM based utterance generation): LLMs have

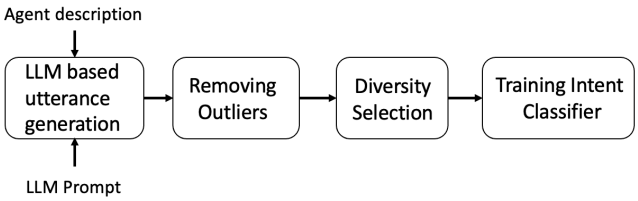

Figure 5: Utterance Generation

shown promising results (Zhao et al. 2023) on various language generation tasks. Using a good prompt with few-shot examples we can use LLMs for text generation without any fine tuning. For utterance generation, we created a text-generation instruction prompt with few-shot examples. As shown in Figure 6.a, prompt comprises of *Input*, *Output* fields and we assign agent description as *Input* and the corresponding human provided utterance for the description as *Output*. A total of 12 *Input*, *Output* pairs are provided as few-shot examples by human experts to limit the number of tokens that can be used as context for LLM inference and therefore the cost. Along with few-shot examples, the prompt contains a *Input* placeholder for the new description and *Output* will be generated by the LLM for the new description.

In this step, we obtain the agent description and add it in the prompt *Input* placeholder for the new description. We then use this prompt for inference on LLM to generate the *Output*. We perform inference on LLM several times using the same prompt to generate multiple utterances. To generate utterances with variations we used LLM inference parameters such as beam-search with sampling for *decoding strategy*, *temperature* is set to a high value (say, 0.9), *maximum new tokens* is set to 20. And for every agent description, we generate 30 utterances via multiple inference requests to the LLM.

```
Generate an Output utterance that may be used to invoke an API using Input sentence
Input: Get the balance of an account
Output: I would like to check the balance of my savings account
...
...
...
Input: {new_description_placeholder}
Output:
```

a) Utterance Generation

```
Generate a Question to request following API Attribute and Description
Description: Buying bus tickets,
Attribute: Source location of the journey,
Question: What is the origin of the bus journey ?
...
...
...
Description:{new_description_placeholder}
Attribute:{new_attribute_placeholder}
Question:
```

b) Question Generation

Figure 6: LLM Prompts

Step-2 (Removing Outliers): Utterances generated using the LLM can be noisy and not relevant. To obtain quality utterances by removing outliers from generated utterances, we consider the agent description as a ground truth seed and use semantic similarity to identify the outliers as below: (i) Obtain sentence embedding[3] features for agent description, generated utterances using model weights `all-MiniLM-L12-v2` (ii) For each generated utterance compute the cosine similarity between agent description and utterance (iii) Remove the utterances that are not semantically similar to the agent description based on the computed cosine similarity score by using the outlier threshold value(say, 0.5). Only utterances with a similarity score above the threshold are selected.

Step-3 (Diversity Selection): After removing outliers from the generated utterances we select a lexically diverse set of utterances as below: i) Obtain a dictionary without English stop words using n-gram vocabulary from the utterances and initialize an empty set for the diverse utterances. ii) For each utterance, compute the n-gram overlap score by counting the presence of vocabulary of the present utterance in the rest of the generated utterances. And select the utterance that has a high n-gram overlap score, and add it to the set of diverse utterances. iii) Iterate over the remaining set of generated utterances, in each iteration compute the n-gram similarity score (Buscaldi et al. 2013) between the current utterance and set of diverse utterances and based on the computed scores the utterance with least n-gram similarity is added to the set of diverse utterances. Also, during each iteration we check for the stopping criteria i.e.; if the size of the diverse utterances is less than the configured limit for the diverse utterances. Upon finishing the iterations a set of diverse utterances are obtained.

Step-4 (Training Intent Classifier): After obtaining diverse utterances from the agent description, we add the utterances as conversational artifacts to the agent specification and update the agents in the database. Finally, we train the intent classifier using agent id as intent class and description, diverse utterances as training samples. In our system, whenever the "builder" persona adds a new API to create an agent, as part of the *build* phase utterance generation is triggered automatically and the intent classifier is trained with augmented training data for the newly created agent. However, the trained intent classifier is used during the *use* phase by the orchestrator for correct agent invocation. Also, the parameters used in different steps of utterance generation such as choice of LLM, LLM inference parameters, number of generated sentences, outlier threshold, n-gram value, and diverse utterance limit can be configured by the "builder".

Question Generation

To make an API service call, it is often necessary to provide certain mandatory attributes as inputs to execute the service. Figure 7 shows an example *Buying bus tickets* API where the service operation */buyBusTickets* requires three fields *from_location, to_location,* and *date* as inputs. While requesting the user to provide inputs for required attributes,

[3]https://www.sbert.net/

```
{
"openapi": "3.0.1",
  "info": { "description": "Buying bus tickets" }
  "servers": [ { "url": "https://prod-server-ticket-reservation.com" }],
  "paths": {
    "/buyBusTickets": {
        "post": {
            "description": "Buying bus ticket request",
            "properties": {
                "from_location": { "title": "Source location for the journey"},
                "to_location": { "title": "Destination location for the journey" },
                "date": { "title": "Date of travel in DD/MM/YYYY format" } },
            "required": ["from_location", "to_location", "date"] } }}
}
```

Figure 7: Example Buying bus tickets API

asking natural language questions based on the attributes will provide an enhanced conversational experience to the user. Examples of such questions can be, *from_location*: What is the origin of the bus journey?, *to_location*: What is the destination of the bus journey? etc., As described in the previous section, LLMs language generation capabilities can be harnessed for the question generation task. The steps to generate questions using LLMs with API meta-data as input are as below:

Step-1 (Required attribute collection): Given an API operation, we extract the list of required attribute names, attributes descriptions from the input schema of the API operation along with the API operation description.

Step-2 (Designing prompt with few-shot examples): For the question generation task we design an instruction prompt with few-shot examples. As shown in Figure 6.b, prompt comprises of *Description*, *Attribute*,*Question* fields and we assign API operation description as *Description*, attribute description (or attribute name if description is not available) as *Attribute* and the corresponding human-provided question for the attribute as *Question*. Our prompt contains 12 few-shot examples annotated by humans similar to the example shown in Figure 6.b. Along with few-shot examples, the prompt contains a *Description* placeholder for the new API operation description, *Attribute* placeholder for the new attribute description and *Question* will be generated by the LLM for the new attribute.

Step-3 (Question generation): Using the list of attributes, prompt from previous steps. For every attribute, we add the attribute description, API operation description in the prompt *Description*, *Attribute* placeholder. And use this prompt for inference on LLM to generate the *Question*. We perform inference with sampling on LLM several times using the same prompt to generate multiple questions. For every attribute, we capture up to three generated questions via multiple inference requests to the LLM.

During the *build* phase, for each agent (API operation), questions are generated for all attributes. Following this, we add the attribute questions as the conversational artifacts to the agent specification and update the agents in the database. The parameters used in different steps of question generation, such as choice of LLM, LLM inference parameters, and

number of questions, can be configured by the "builder".

Next Best Agent Recommendation

In the *build* phase of an agent (which is referred to as the *target_agent*), we compute the NBA(Next Best Agents) that are probable candidates to be executed after the *target_agent*. In our system, we recommend the top-k NBAs for the *target_agent* to provide a better conversational experience to the user. NBA component of the system at first, extracts data such as description, input, and output signatures from all available agents in the agent database. Using the extracted data we create a semantic search index for the agents. The purpose of creating an index for the agents is to enable relevance search on all available agents according to the query i.e.; *target_agent* description. For enabling semantic search we use ChromaDB[4], which is a vector database that uses `all-MiniLM-L12-v2` model weights to create a search index using the sentence embedding of the agent description as features. Below we present the algorithm steps to compute NBAs using semantic search index and input-output signature similarity.

Step-1 (Semantic Pruning): ChromaDB-based semantic search index supports querying to get relevant APIs from all available agents (S) and we use the cosine similarity to find the most similar top-k agents from S to the *target_agent*

Step-2 (Input-Output Match Ranking): When agents are sequenced, the data flows from one agent to the next one. Therefore, in this step, we take the top-k agents obtained from Step-1 and rank them based on whether the output of the *target_agent* matches with the input of the candidate NBA agent or not. We provide the final ranking of top-k agents based on the computed input-output matching score.

- *Calculating Input-Output matching score:* We obtain the sentence embeddings of the concatenated attribute descriptions in both the output signature of *target_agent* and the input signature of each candidate NBA agent. We then calculate cosine similarity and use it as an Input-Output matching score in Step-2.

After obtaining NBAs for the *target_agent*, we add them as conversational artifacts to the agent specification and update the agent specification in the database. The parameters used in different steps of NBA, like the choice of embedding model and the value of top-k can be configured by the "builder" persona.

Validation

In this section, we present the validation results on the different components of the system.The validation is done without considering the impact of the modifications of the build persona. In practice, the build persona may or may not modify the conversational artifacts resulting in a higher system performance. Moreover, the Watson Orchestrate product also features systems for online learning where the intent classifier and Next best actions are trained with historical client-specific usage data, the impact of these is not considered here. Following are the datasets used for validation:

- **SGD**: Schema Guided Dialogue dataset (Rastogi et al. 2020) consists of conversations between a virtual assistant and a user from several domains including Travel, Events, Payment, Media, Restaurants, Weather, etc. It includes annotations for natural language understanding, dialogue state tracking, API service schema, and user utterances for API invocation
- **HR**: Human Resource(HR) dataset is an internal dataset, created from the APIs used in various HR operations and the deployed usage data of these APIs. It includes human annotations for the user utterances to invoke API, natural language understanding, and NBA (Next Best Agent)
- **ABCD**: Action Based Conversational dataset (Chen et al. 2021), consists of human-to-human dialogues related to user intents along with sequences of actions to achieve a task. It includes annotations for user utterances to invoke an action, natural language understanding, and NBA(Next Best Agent)

Validation of Utterance Generation

The Utterance Generation component is used in *build* phase for augmenting training data, training an intent classifier (which is part of the *understand* skill used by the orchestrator). To validate the performance of the Utterance Generation component, we evaluate the intent classifier and show the value of utterance generation for improving intent classification. For intent classifier evaluation, we considered two different intent classification algorithms, (i)Watson Assistant(WA)[5], and (ii) XGBoost [6]. And intent classifiers are trained and tested on two datasets (i)HR and (ii)SGD with utterance generation using different LLMs. **Datasets**: HR dataset consists of 22 APIs and for the intent classification task we processed the API specifications to obtain the required data, which consists of 22 API descriptions and 482 human annotated utterances to invoke the API. Similarly, for the SGD (Rastogi et al. 2020) dataset we processed the dialogues from the test-set split and extracted the first turn of each dialogue to obtain user utterance along with associated API. Processed SGD test-set consists of 26 API descriptions and 4201 user utterances. In both datasets, API descriptions are the agent description and user utterances are used as test samples for intent classification.

LLMs: We evaluate three different open source LLMs, FLAN-T5-XXL (Chung, Hou, and et. al. 2022), FLAN-UL2 (Tay et al. 2023), Llama2-13b (Touvron et al. 2023) for training data augmentation

Metrics: Intent classifier performance is evaluated using these metrics, (i)*Fallback*: The ratio of the number of Fallback predictions and the total number of predictions, (ii)*Accuracy*: The ratio of the number of correct predictions and the total number of predictions, (iii)*F1-score*: The harmonic mean of Precision and Recall and the (iv)*In-correct*: The ratio of number of predictions that are neither *correct* nor *Fallback* and the total number of predictions. It represents the actual misclassification rate.

[4]https://github.com/chroma-core/chroma

[5]https://www.ibm.com/products/watson-assistant

[6]https://xgboost.readthedocs.io/en/stable/python/

In our evaluation, each of the intent classifier algorithms is trained in four variations i.e.; the Baseline model uses only descriptions as training samples and the other three variations use different LLMs for generating utterances to augment the training data. The intent classifiers are trained using WA which is a proprietary algorithm and provides SDK(software development kit) for train and predict functionalities where whereas XGBoost is an open-source algorithm trained using tf-idf (Ramos et al. 2003) features with 5-fold cross-validation to find optimal hyper-parameters. We use different variations of trained intent classifiers to predict the agent for the test user utterances, if the model prediction probability score is lower than the confidence threshold it will predict the intent class as 'Fallback' i.e.; Orchestrator will not select an agent. It is important to assign a valid threshold for 'Fallback' and the optimal value for the threshold is determined during the classifier training using 5-fold cross-validation. Validation results are shown in Table 1, for both the classifier algorithms the LLM utterance generation results are better than the Baseline in terms of *Accuracy*, *F1-score*, *In-correct* metrics.

Validation of Question Generation

The Question Generation component is evaluated on the SGD (Rastogi et al. 2020) dataset. This dataset contains dialogues with API schema attributes associated with user questions in conversations between a virtual assistant and an user. We processed the test-set split of the dataset to obtain the API service description, schema attribute description, and associated questions asked by the virtual assistant. Through human annotations, we hand-picked only the samples with high quality. In this curated test data there are 128 sample records with *Description*,*Attribute*,*Question*. We evaluated several LLMs such as FLAN-T5 (Chung, Hou, and et. al. 2022), FLAN-UL2 (Tay et al. 2023), Llama2-13b (Touvron et al. 2023) to generate questions on the test data.

As described in section *Question Generation*, for each sample in the test set, we obtain the few-shot examples prompt with API description and attribute description populated in the *Description*,*Attribute* placeholders. Using this prompt we perform inference on the LLM to generate the *Question* for the given attribute. To evaluate the quality of the generated questions, they need to be compared with the ground truth questions in the curated dataset. Of course, the generated questions do not have to exactly match, but we like them to be semantically similar. The ideal way to make this comparison would be for human experts to annotate the results, indicating whether each of the questions is valid/usable or not. However, due to the large number of samples, annotation of all samples is cumbersome and not viable.

Our approach is to estimate the semantic similarity between the generated question GQ_i and the ground truth question Q_i for each sample i. Using sentence embedding[7] of the questions from the `all-MiniLM-L12-v2` model weights, and the cosine similarity is computed (denoted as $S(GQ_i, Q_i)$). While the similarity between the generated and ground truth questions can be computed, it still does not indicate if the questions were usable.

To alleviate this issue, a subset of the generated questions (around 15%) were annotated by human experts with each sample, indicating whether the generated question was usable or not. This annotation is used to calibrate a model-specific threshold λ on the similarity score. Any generated question for which $S(GQ_i, Q_i) > \lambda$ is deemed to be good. The optimal threshold λ is obtained based on k-fold cross-validation on the annotated subset to best match the human annotation.

Once the similarity scores are calculated for all 128 test samples with that of the ground truth questions, we compute the fraction of cases for which the generated questions have a similarity score greater than the threshold λ. This is referred to as the accuracy of question generation (QG-Accuracy). This also reflects the effectiveness of the underlying LLM for the task of question generation.

Table 2 shows the QG-accuracy of various LLMs. It can be observed that the questions generated through Llama-2-13b-Chat, FLAN-T5-XL and FLAN-UL2 seem to be the best. Also, as we take a larger LLM in terms of the number of parameters, the accuracy does not always improve. Interestingly, the smaller FLAN-T5-XL model in terms of parameters (3B) produces a better result than the FLAN-T5-XXL (11B) which may be an important consideration in terms of cost deployment.

Validation of Next Best Agents

Our proposed NBA(Next Best Agents) recommendation algorithm is validated using two datasets: HR and ABCD. In the HR dataset, we have 22 APIs/Agents with the description and NBA annotations. For the ABCD dataset, we processed the raw data to obtain 30 different actions and their NBAs.
Metrics: To evaluate the performance of NBA, we define two evaluation metrics, i)*NBA Accuracy*: The fraction of total number of NBAs predicted correctly and the total number of NBAs present in the Ground Truth (GT) across all agents, ii)*Agent Recall*: The fraction of number of agents for which at least one NBA was predicted correctly in top-k out of the total number of recommended agents.

We evaluate the performance of our approach using Elastic Search based on the baseline and it supports Keyword based search. In the Elastic Search baseline, we first obtain features using BM25 (Robertson and Zaragoza 2009) method for all the agent descriptions, and the search index is created using the BM25 features. This Keyword based search index supports querying the relevant top-k APIs based on API description and the ranking of APIs is computed using the cosine distance between tf-idf (Ramos et al. 2003) features of *target_agent* and all agents in the index. We use $k = 5$ in our experiments and the results obtained from our proposed approach are shown in Table 3. As shown, our approach outperforms the Elastic Search baseline on the ABCD dataset and performs comparably on the HR dataset. This is because the API descriptions in the HR dataset contain high keyword overlap that Elastic Search is able to handle in a better way. However, our approach would scale better for more natural agent descriptions.

[7] https://www.sbert.net/

Classifier	LLM	HR				SGD			
		Accuracy	*Incorrect*	*Fallback*	*F1*	*Accuracy*	*Incorrect*	*Fallback*	*F1*
WA	Baseline	0.53	0.28	0.19	0.53	0.35	0.55	0.10	0.49
WA	FLAN-T5-XXL	**0.66**	**0.27**	0.07	**0.67**	**0.71**	**0.15**	0.14	**0.72**
WA	FLAN-UL2	0.59	0.31	0.10	0.61	0.68	0.17	0.15	0.67
WA	Llama2-13b	0.65	0.29	0.06	0.65	0.68	0.18	0.14	0.71
XGBoost	Baseline	0.04	0.86	0.10	0.02	0.02	0.76	0.22	0.02
XGBoost	FLAN-T5-XXL	0.52	0.40	0.08	0.50	0.51	0.46	0.03	0.53
XGBoost	FLAN-UL2	0.49	0.47	0.04	0.48	0.43	0.39	0.18	0.47
XGBoost	Llama2-13b	0.46	0.45	0.09	0.45	0.51	0.35	0.14	0.53

Table 1: Intent Classification Results

LLM	QG Accuracy (SGD)
FLAN-T5-XL	0.734
FLAN-T5-XXL	0.719
FLAN-UL2	0.742
Llama-2-13b	0.609
Llama-2-13b-Chat	**0.797**

Table 2: Accuracy of Question Generation as Compared to Ground-truth Questions

Method	NBA Accuracy		Agent Recall	
	ABCD	*HR*	*ABCD*	*HR*
Our Approach	**0.26**	0.39	**0.50**	**0.61**
Elastic Search	0.10	**0.40**	0.27	0.58

Table 3: Next Best Agent Results

Deployments Details

Our system is deployed and used as part of the Watson Orchestrate[8] product with general availability to the customers since December 2022, also the product is honored with CES Innovation Award 2022[9]. To date, the product has been deployed in over 2500 tenants and used by more than 11000 active users. Over 600 APIs have been wrapped up as agents by *builders*. Further, these agents have been invoked over 220,000 times by *users*. In this paper, we evaluated the system of Watson Orchestrate focused on HR process use-case with APIs related to candidate sourcing, career mobility, human resources, onboarding, procurement, recruiting, and talent acquisition. The product offers out-of-the-box support for several business-critical APIs and RPAs services with a conversational interface enabled. The services include Asana, Dropbox, Oracle HC, Salesforce, SAP SuccessFactors, IBM RPA, ThisWay Global, Jira, Slack, Gmail, etc.,

For the system development, we used Python for the backend components deployed on the Openshift cloud platform[10] as a multi-tenant service. Openshift platform provisions different hardware requirements for various components. For deploying the LLMs we leverage the NVIDIA A100 GPUs and for the other components, CPUs are sufficient. We use asynchronous inference APIs using FastAPI[11] framework for different LLMs such that all the other components of the system can make REST API calls to the LLM inference service. The other components of the system are packaged as different docker images with each handling different functionalities such as Utterance Generation, Question Generation, NBA, Orchestrator, and Database. In the deployment system components can be configured and managed independently from each other. In our internal analysis, we have observed a 35-minute reduction in the build effort per skill, that corresponds to a 77% reduction in build effort. We use an automated DevOps pipeline for system deployment and maintenance. Maintenance includes deploying new releases for the system components with enhancements and bug fixes from product usage. Parameters such as, choice of LLM and inference parameters, can be set up through a configuration menu in the UI. Not all builders would need to access this or have a knowledge of LLMs. These may be configured once for a tenant.

Conclusions and Future Work

Generating conversational artifacts such as sample utterances for data augmentation, question generation for slot-filling and next action predictions can simplify the process of creating chatbots and improving the user experience, particularly for those without the necessary AI expertise. Our methods demonstrated their effectiveness by producing significant accuracy improvements across various datasets. These methods were also successfully integrated into Watson Orchestrate Product. Nevertheless, there are limitations to this approach that we intend to tackle in our future endeavors. Specifically, we would like to utilize human-in-the-loop validation to enhance our methods and capitalize on end-user feedback to further enrich the dataset and facilitate model retraining. The performance of the NBA can further be improved by leveraging the process knowledge and the usage data collected from the proposed system.

References

Bang, Y.; Cahyawijaya, S.; Lee, N.; Dai, W.; Su, D.; Wilie, B.; Lovenia, H.; Ji, Z.; Yu, T.; Chung, W.; Do, Q. V.; Xu,

[8]https://www.ibm.com/products/watson-orchestrate

[9]https://www.ces.tech/innovation-awards/honorees/2022/honorees/w/watson-orchestrate.aspx

[10]https://www.redhat.com/en/technologies/cloud-computing/openshift

[11]https://fastapi.tiangolo.com/

Y.; and Fung, P. 2023. A Multitask, Multilingual, Multimodal Evaluation of ChatGPT on Reasoning, Hallucination, and Interactivity. arXiv:2302.04023.

Bocklisch, T.; Faulkner, J.; Pawlowski, N.; and Nichol, A. 2017. Rasa: Open source language understanding and dialogue management. *arXiv preprint arXiv:1712.05181*.

Buscaldi, D.; Le Roux, J.; García Flores, J. J.; and Popescu, A. 2013. LIPN-CORE: Semantic Text Similarity using n-grams, WordNet, Syntactic Analysis, ESA and Information Retrieval based Features. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, 162–168. Atlanta, Georgia, USA: Association for Computational Linguistics.

Chen, D.; Chen, H.; Yang, Y.; Lin, A.; and Yu, Z. 2021. Action-Based Conversations Dataset: A Corpus for Building More In-Depth Task-Oriented Dialogue Systems. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021*, 3002–3017. Online: Association for Computational Linguistics.

Chung, H. W.; Hou, L.; and et. al., L. 2022. Scaling Instruction-Finetuned Language Models. *arXiv preprint arXiv:2210.11416*.

Devarajan, Y. 2018. A study of robotic process automation use cases today for tomorrow's business. *International Journal of Computer Techniques*, 5(6): 12–18.

Laurent, P.; Chollet, T.; and Herzberg, E. 2015. Intelligent automation entering the business world. *Deloitte, available at https://www2. deloitte. com/content/dam/Deloitte/lu/Documents/operations/lu-intelligent-automationbusiness-world. pdf (accessed 5th March, 2018)*.

Patil, S. G.; Zhang, T.; Wang, X.; and Gonzalez, J. E. 2023. Gorilla: Large Language Model Connected with Massive APIs. *arXiv preprint arXiv:2305.15334*.

Ramos, J.; et al. 2003. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, 29–48.

Rastogi, A.; Zang, X.; Sunkara, S.; Gupta, R.; and Khaitan, P. 2020. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 8689–8696.

Rizk, Y.; Isahagian, V.; Boag, S.; Khazaeni, Y.; Unuvar, M.; Muthusamy, V.; and Khalaf, R. 2020. A Conversational Digital Assistant for Intelligent Process Automation. In *BPM 2020 Blockchain and RPA Forum*, volume 393 of *LNBIP*, 85–100.

Robertson, S.; and Zaragoza, H. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval*, 3: 333–389.

Sabharwal, N.; Barua, S.; Anand, N.; and Aggarwal, P. 2020. *Building Your First Bot Using Watson Assistant*, 47–102. Berkeley, CA: Apress. ISBN 978-1-4842-5555-1.

Syed, R.; Suriadi, S.; Adams, M.; et al. 2020. Robotic Process Automation: Contemporary themes and challenges. *Comput. Ind.*, 115: 103162.

Tay, Y.; Dehghani, M.; Tran, V. Q.; Garcia, X.; Wei, J.; Wang, X.; Chung, H. W.; Shakeri, S.; Bahri, D.; Schuster, T.; Zheng, H. S.; Zhou, D.; Houlsby, N.; and Metzler, D. 2023. UL2: Unifying Language Learning Paradigms. arXiv:2205.05131.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; et al. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288.

Vaziri, M.; Mandel, L.; Shinnar, A.; Siméon, J.; and Hirzel, M. 2017. Generating chat bots from web API specifications. In *Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, 44–57.

Yaghoub-Zadeh-Fard, M.-A.; Benatallah, B.; and Zamanirad, S. 2020. Automatic Canonical Utterance Generation for Task-Oriented Bots from API Specifications. In *EDBT*.

Yang, Y.; Uy, M. C. S.; and Huang, A. 2020. Finbert: A pretrained language model for financial communications. *arXiv preprint arXiv:2006.08097*.

Zhao, W. X.; Zhou, K.; Li, J.; Tang, T.; Wang, X.; Hou, Y.; Min, Y.; Zhang, B.; Zhang, J.; Dong, Z.; Du, Y.; Yang, C.; Chen, Y.; Chen, Z.; Jiang, J.; Ren, R.; Li, Y.; Tang, X.; Liu, Z.; Liu, P.; Nie, J.-Y.; and Wen, J.-R. 2023. A Survey of Large Language Models. arXiv:2303.18223.